A Project Report On

Statistical Models and Their Applications in Natural Language Processing

Submitted in partial fulfillment of the requirement for the 3^{rd} semester

Master of Science

in

STATISTICS

DEPARTMENT OF STATISTICS,

SCHOOL OF MATHEMATICAL SCIENCES,

KAVAYITRI BAHINABAI CHAUDHARI NORTH MAHARASHTRA UNIVERSITY

'A' Grade NAAC Re-accredited (4^{th} Cycle)

Umavi Nagar, Jalgaon - 425001 (M.S.) India

Submitted By

MARATHE HARSHAL BALKRUSHNA

Seat no. **366346**

Under the guidance of

Mr. MANOJ C. PATIL

Asst. Professor, dept. of Statistics , K.B.C N.M.U.

December 2024

KAVAYITRI BAHINABAI CHAUDHARI NORTH MAHARASHTRA UNIVERSITY

SCHOOL OF MATHEMATICAL SCIENCES

Re-accredited by National Assessment & Accreditation Council (NAAC) with 'A' grade 4th Cycle)

Umavi Nagar, Jalgaon - 425001 (M.S.) India

DEPARTMENT OF STATISTICS

CERTIFICATE

This is to certify that the project entitled **Statistical Models and Their Applications in Natural Language Processing** is a bonafide work carried out by **HARSHAL BALKRUSHNA MARATHE [366346]** in partial fulfillment of 3rd semester, Master of Science in STATISTICS under Kavayitri Bahinabai Chaudhari University North Maharashtra University, Jalgaon during the year 2024-25.

Mr. Manoj C. Patil
(Project Guide)
Asst Prof. Dept. of Statistics, KBCNMU

Signature:......................

Acknowledgement

I am pleased to have successfully completed the project **Statistical Models and Their Applications in Natural Language Processing**. I thoroughly enjoyed the process of working on this project and gained a lot of knowledge doing so.

I would like to take this opportunity to express my gratitude to **Prof. (Mrs.) K. K. Kamalja** , Head of Department, Department of Statistics, for permitting me to utilize all the necessary facilities of the institution.

I am immensely grateful to my respected and learned guide, **Mr. Manoj C. Patil**, Assistant Professor, Dept. of Statistics for his valuable help and guidance. I am indebted to him for his invaluable guidance throughout the process and his useful inputs at all stages of the process.

I would also like to thank **Prof. Dr. R. L. Shinde** and **Prof. R. D. Koshti** for their help and guidance throughout the project process.

I also thank all the faculty and support staff of Department of Statistics, School of Mathematical Sciences. Without their support , this work would not have been possible.

Lastly, I would like to express my deep appreciation towards my classmates and my family for providing me with constant moral support and encouragement. They have stood by me in the most difficult of times.

Harshal Marathe (366346)

Abstract

Natural Language Processing (NLP) has become a cornerstone of modern computational linguistics, enabling machines to understand and interact with human language. Statistical models play a pivotal role in the development and improvement of NLP systems by providing robust frameworks for language understanding. This project explores the fundamental statistical models used in NLP, including N-grams, Hidden Markov Models (HMMs), Naive Bayes classifiers, and their applications in various NLP tasks such as text classification, sentiment analysis, and machine translation. Through empirical analysis, we examine how these models are trained, evaluated, and applied to real-world datasets, showcasing their strengths and limitations. Additionally, the project delves into the challenges encountered when using statistical models in NLP, including data sparsity, ambiguity, and model interpretability. The results highlight the potential of statistical models in NLP while offering insights into future directions and improvements. This work aims to provide a comprehensive understanding of the theoretical foundations and practical applications of statistical methods in NLP, with a focus on their impact on language processing tasks across diverse domains.

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Background

1.1.1 Natural Language Processing (NLP)

Natural Language Processing (NLP) is a field of artificial intelligence that enables computers to understand, interpret, and respond to human language. It encompasses various techniques for transforming unstructured text data into a structured format that can be analyzed, which is critical for applications ranging from simple text analytics to sophisticated dialogue systems.

- **Importance of NLP in Data Science and Machine Learning**
  - **Textual Data as a Resource**: With the increasing availability of textual data through news articles, social media, and other online platforms, NLP has become a cornerstone of data science. Text data can provide insights into customer preferences, public opinions, and market trends, making it an invaluable resource across industries.

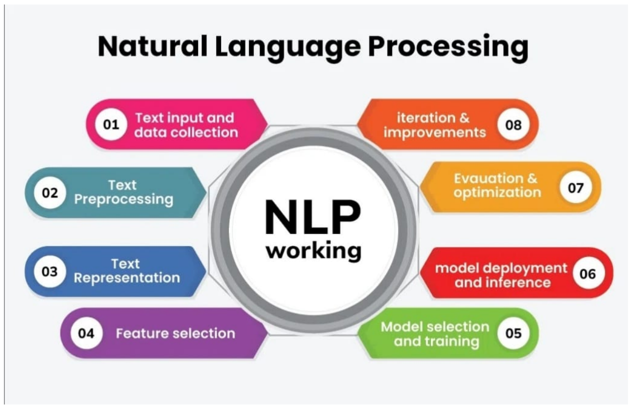
Natural Language Processing
NLP working
01 Text input and data collection
02 Text Preprocessing
03 Text Representation
04 Feature selection
05 Model selection and training
06 model deployment and inference
07 Evauation & optimization
08 iteration & improvements

Figure 1.1: NLP.

- **Core Tasks in NLP**: NLP includes various tasks, such as tokenization, stemming, and lemmatization, which prepare text for further analysis. Additionally, advanced tasks like sentiment analysis, named entity recognition, and topic modeling can extract specific insights and patterns.

- **Advancements in NLP Models**: Over the years, NLP has evolved from basic keyword matching techniques to advanced deep learning models like RNNs, LSTMs, and transformer-based models (e.g., BERT, GPT). These models have vastly improved the accuracy of language processing tasks and enabled machines to understand context and semantics.

Applications of NLP in Various Domains

1. **Text and Document Analysis**
    - *Sentiment Analysis*: Determining the sentiment of a text, such as positive, negative, or neutral (e.g., social media monitoring, customer feedback analysis).
    - *Text Summarization*: Extracting the most important information from a large text or document.
    - *Topic Modeling*: Identifying underlying topics in large datasets (e.g., news articles, reviews).
    - *Named Entity Recognition (NER)*: Extracting specific entities like names, locations, dates, and organizations.
2. **Language Translation**
    - *Machine Translation*: Translating text from one language to another (e.g., Google Translate).
    - *Localization*: Adapting content for different languages and regions, including cultural nuances.
3. **Search and Information Retrieval**
    - *Search Engines*: Enhancing search results with query understanding and semantic search (e.g., Google, Bing).
    - *Chatbot Question Answering*: Retrieving the most relevant information from a database or knowledge base (e.g., FAQs).
4. **Virtual Assistants**
    - *Voice Assistants*: Enabling interactions with devices using speech (e.g., Siri, Alexa, Google Assistant).
    - *Chatbots*: Conversational agents for customer service, e-commerce, and support.
5. **Customer Interaction**
    - *Speech-to-Text and Text-to-Speech*: Converting spoken language to written text and vice versa.

- *Personalized Recommendations*: Recommending products or services based on user reviews and interactions.

6. **Healthcare**

   - *Medical Record Analysis*: Analyzing clinical notes for insights (e.g., extracting symptoms, diagnoses).
   - *Drug Discovery*: Analyzing research papers for relationships between drugs, diseases, and proteins.
   - *Patient Chatbots*: Assisting patients with medical queries or scheduling appointments.

7. **Financial Services**

   - *Sentiment Analysis for Market Predictions*: Analyzing news and social media sentiment for stock market insights.
   - *Fraud Detection*: Identifying anomalies in transaction patterns using language data.
   - *Document Parsing*: Automating data extraction from financial statements and contracts.

1.1.2 Background of Statistics in NLP

The integration of statistics into NLP emerged in the late 20th century as a response to the limitations of earlier rule-based systems. Early NLP systems relied on handcrafted grammar rules, which were labor-intensive, domain-specific, and lacked scalability. The advent of computational power and the availability of large text corpora led to the rise of statistical and machine learning methods, transforming NLP into a data-driven discipline.

Key Developments in Statistics and NLP

1. **1950s–1970s: Early Experiments with Probabilistic Models**

   - Early models like Shannon's work on information theory laid the groundwork for probabilistic language modeling.
   - Techniques like Markov Chains and Hidden Markov Models (HMMs) were introduced to model sequences of words or phonemes, forming the basis of speech recognition and simple text prediction.

2. **1980s–1990s: Emergence of Statistical NLP**

   - Statistical methods became mainstream with the introduction of corpora such as the Brown Corpus.
   - *n-gram* language models and Maximum Likelihood Estimation (MLE) were used for text generation and word prediction.

- Probabilistic approaches like Naive Bayes, Logistic Regression, and Latent Semantic Analysis (LSA) were applied to tasks such as sentiment analysis, topic modeling, and document classification.

3. **2000s: Machine Learning in NLP**

   - Support Vector Machines (SVMs), Decision Trees, and Random Forests were used to tackle NLP tasks like spam detection, named entity recognition, and part-of-speech tagging.
   - Statistical methods evolved to include unsupervised learning techniques like clustering (e.g., *k-means*) and dimensionality reduction (e.g., PCA, t-SNE).

4. **2010s–Present: Neural Networks and Deep Learning**

   - While traditional statistical methods still play a crucial role, the advent of deep learning has introduced powerful models like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers.
   - These models rely on large-scale datasets and are optimized using statistical techniques such as gradient descent, cross-entropy loss, and regularization.

1.2 Objective

The primary objective of this project is to explore, implement, and evaluate various statistical methods in Natural Language Processing (NLP) to understand their role in analyzing and processing human language. The project aims to investigate how statistical techniques enable machines to interpret, model, and generate natural language data effectively.

1. **Understanding Statistical Foundations in NLP**

   - To study and understand the statistical principles and probabilistic models that form the backbone of NLP applications.
   - To explore fundamental concepts like probability distributions, Markov models, and Bayes' theorem in the context of language processing.

2. **Modeling Language with Statistical Techniques**

   - To build and evaluate language models, such as *n-gram* models, Hidden Markov Models (HMMs), and Latent Dirichlet Allocation (LDA), for tasks like language generation, part-of-speech tagging, and topic modeling.
   - To assess the performance of these models using metrics like perplexity, log-likelihood, and accuracy.

3. **Feature Engineering and Text Representation**

   - To investigate statistical methods for feature extraction and dimensionality reduction, including term frequency-inverse document frequency (TF-IDF), word embeddings, and Principal Component Analysis (PCA).
   - To analyze the impact of feature representation on model performance.

4. **Application of Statistical Methods**
   - To apply statistical models to solve real-world NLP tasks such as sentiment analysis, document classification, and spam detection.
   - To compare statistical approaches with modern deep learning techniques for similar tasks.

5. **Performance Evaluation and Optimization**
   - To evaluate the strengths and limitations of different statistical models across various NLP tasks.
   - To explore techniques for improving model robustness, scalability, and efficiency in processing large-scale language data.

6. **Insights and Advancements**
   - To derive actionable insights on how statistical methods contribute to NLP advancements and their synergy with deep learning methods.
   - To provide recommendations on the suitability of statistical models for specific NLP tasks and datasets.

1.3 Scope of the Project

The scope of this project, *Statistics in NLP*, encompasses the theoretical and practical exploration of statistical methods to solve a wide range of language processing tasks. By focusing on the application of statistical models, the project aims to bridge the gap between foundational concepts and real-world applications in Natural Language Processing.

Key Areas Covered

1. **Theoretical Foundation**
   - Detailed study of statistical principles such as probability theory, Markov models, and Bayes' theorem.
   - Exploration of the role of statistics in language modeling, text representation, and feature extraction.

2. **Statistical Models in NLP**
   - Development and implementation of key statistical models, including:
     - *n-Gram Models* for language modeling and text prediction.

  - *Hidden Markov Models (HMMs)* for sequence analysis and part-of-speech tagging.
  - *Naive Bayes* for text classification and sentiment analysis.
  - *Latent Dirichlet Allocation (LDA)* for topic modeling.
- Comparative analysis of these models to assess their performance, efficiency, and robustness.

3. **Feature Engineering and Representation**
   - Use of statistical methods like TF-IDF, word frequency analysis, and dimensionality reduction (e.g., PCA) to preprocess and represent text data.
   - Analysis of how these techniques impact the overall effectiveness of NLP models.
4. **Applications of Statistical NLP**
   - Implementation of statistical techniques for solving common NLP tasks, such as:
     - Sentiment analysis.
     - Document classification.
     - Named Entity Recognition (NER).
   - Real-world applications, including spam detection, topic discovery, and customer feedback analysis.
5. **Performance Evaluation**
   - Assessment of statistical models using metrics such as accuracy, perplexity, F1-score, and log-likelihood.
   - Comparison with modern deep learning approaches to highlight the strengths and limitations of statistical methods.
6. **Insights and Advancements**
   - Exploration of the relevance of statistical methods in modern NLP applications.
   - Identification of opportunities for integrating statistical approaches with advanced machine learning and deep learning models.

1.4 Literature Review: Statistics in NLP

The use of statistical methods in Natural Language Processing (NLP) has revolutionized how machines understand and generate human language. This shift from rule-based systems to data-driven models has enabled more accurate and scalable solutions for processing text.

Daniel Jurafsky and James H. Martin Jurafsky and Martin (2023)
The book Speech and Language Processing by Jurafsky and Martin is a foundational text in the fields of natural language processing (NLP), computational linguistics, and speech recognition. As the third edition draft comprehensively addresses modern advancements in NLP, it is particularly relevant for projects involving sentiment analysis, text representation, and predictive modeling, such as NLP-driven Stock Movement Prediction Using News Sentiment. Below, key aspects of the book are reviewed with a focus on their relevance to your project:

1. **Foundations of Natural Language Processing**
   The book begins with an introduction to the principles of language modeling, tokenization, and text preprocessing, which are crucial for preparing unstructured text data. It delves into methods such as n-grams, stemming, and lemmatization, laying the groundwork for transforming raw news data into analyzable formats. These preprocessing techniques directly support your goal of using past news for stock movement prediction by ensuring consistent and structured data representation.

2. **Sentiment Analysis**
   A significant section of the book explores sentiment analysis, particularly the use of machine learning methods to classify text polarity. Jurafsky and Martin provide detailed explanations of feature engineering, lexical resources (e.g., sentiment lexicons), and classification algorithms (Naive Bayes, SVMs, etc.). This aligns closely with the sentiment scoring aspect of your project, where news sentiment is analyzed and correlated with stock price trends.

3. **Word Representations and Embeddings**
   The authors address modern text representation techniques such as word embeddings (Word2Vec, GloVe) and contextual embeddings (ELMo, BERT). These techniques enhance the ability to capture semantic nuances in news articles, improving the model's understanding of market sentiment and its implications. Leveraging embeddings can help your LSTM model better associate linguistic patterns with stock price movements.

4. **Sequence Modeling with RNNs and LSTMs**
   Jurafsky and Martin discuss recurrent neural networks (RNNs) and their improved variant, long short-term memory networks (LSTMs), which are central to your stock movement prediction model. Their explanation includes sequence modeling for tasks like text generation, time-series prediction, and speech recognition. These insights are invaluable for implementing an LSTM-based architecture to predict stock movements based on sequential news and price data.

Nivre (2001)
On Statistical Methods in Natural Language Processing," Joakim Nivre explores the increasing use of statistical approaches in NLP, highlighting their advantages over traditional rule-based methods. The paper discusses how statistical models, such as Hidden Markov Models (HMM) and N-gram models, have become central to language processing tasks by providing a probabilistic framework for modeling linguistic phenomena. Nivre emphasizes the importance of large corpora for training these models and the role of machine learning techniques in improving their performance. The paper also addresses challenges related to data sparsity and model complexity. Ultimately, Nivre advocates for the integration of statistical methods into NLP systems to handle the variability and ambiguity inherent in human language

Berg-Kirkpatrick et al. (2012)
"An Empirical Investigation of Statistical Significance in NLP," Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein investigate the role of statistical significance in evaluating NLP models. The authors analyze various techniques for assessing the performance of statistical models in natural language processing, highlighting the limitations of traditional significance tests in this domain. They argue that understanding statistical significance is crucial for ensuring that improvements in NLP systems are not due to random chance. The paper provides empirical evidence and proposes alternative methods for evaluating the reliability of NLP results, offering insights into best practices for comparing model performance. The findings emphasize the importance of robust statistical analysis to interpret and validate NLP research outcomes effectively.

Jacobs (1992) "Joining Statistics with NLP for Text Categorization," Paul S. Jacobs explores the integration of statistical methods with natural language processing techniques to improve text categorization. The paper emphasizes the importance of statistical approaches in automating the classification of large text corpora, presenting methods for analyzing text features such as word frequencies and patterns. Jacobs demonstrates how combining traditional NLP techniques with statistical models can enhance the accuracy and efficiency of text categorization tasks. The work laid the foundation for future developments in statistical machine learning for NLP, showing the potential for applying statistical methods to real-world text classification problems.

Chapter 2

Statistical Analysis

2.1 Data Preprocessing

Data preprocessing is a crucial step in preparing raw text data for input into RNN models. Proper preprocessing ensures that the data is in a suitable format and cleansed of unnecessary noise, which can significantly improve model performance. The following outlines the steps involved in data preprocessing for text generation using RNNs.

2.1.1 Text Preprocessing

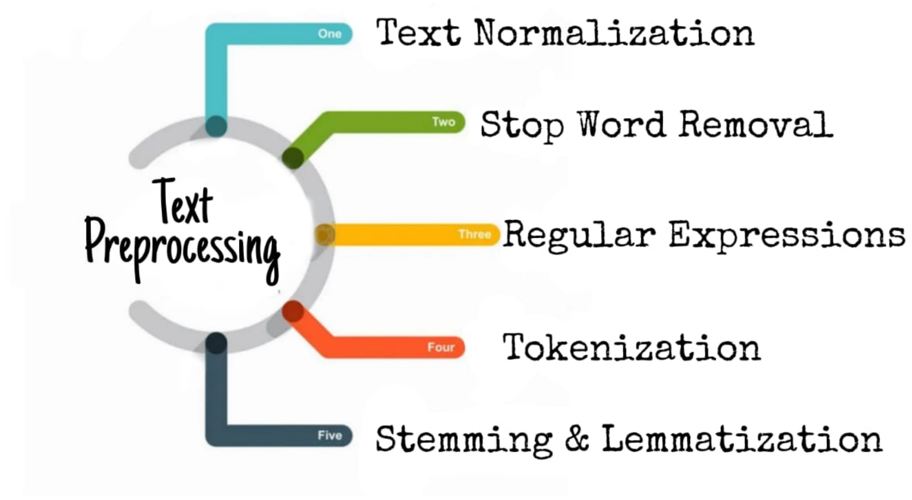

Figure 2.1: Text Preprocessing.

1. **Text Cleaning** Text cleaning involves removing unwanted elements and standardizing the text to ensure consistency across the dataset.

    - Lowercasing: Convert all text to lowercase to reduce the vocabulary size and ensure uniformity.
    - Removing Punctuation and Special Characters: Strip out punctuation marks, special characters, and numbers that do not contribute to text generation

    Appendix A.1.1

2. **Tokenization** Tokenization is the process of splitting the text into smaller units (tokens) such as words or characters. This is a key step for converting text into numerical form that can be processed by RNNs.

    - Word-Level Tokenization: Splits text into words, useful for models generating word sequences.
    - Character-Level Tokenization: Splits text into characters, which is useful for finer control in text generation, especially when handling unknown words or creative text generation.

    A.1.2

3. **Stemming and lemmatization** Stemming and lemmatization are two techniques used in natural language processing (NLP) to reduce words to their root form. While both aim to normalize text, they approach the task in different ways.

    Stemming

    - **Definition** : Stemming is a process of removing suffixes from words to reduce them to their root or stem form.
    - **Approach**: Stemming is typically rule-based, using a set of predefined rules to identify and remove suffixes.
    - Example: Stemming: "running" -> "run" Issue: Stemming can sometimes produce incorrect results, as it doesn't consider the word's context or part of speech. For example, "run" can also be the base form of "ran" and "runs."

    Appendix A.1.3

    Lemmatization

    - **Definition**: Lemmatization is a more sophisticated process that converts a word to its base form, considering its part of speech and morphological rules.
    - **Approach**: Lemmatization often involves using a dictionary or lexicon to look up the base form of a word.
    - Example: Lemmatization: "better" -> "good" Advantage: Lemmatization produces more accurate results than stemming, as it takes into account the word's context and meaning.

Appendix A.1.4

Key Differences:

Feature	**Stemming**	**Lemmatization**
Approach	Rule-based	Dictionary-based
Accuracy	Less accurate	More accurate
Context	Ignores context	Considers context
Output	Stem form	Base form

When to Use Which:

Stemming: Suitable for simple tasks where accuracy isn't critical, such as keyword extraction or information retrieval.
Lemmatization: Recommended for tasks where accuracy is important, such as sentiment analysis or text classification.

2.1.2 Text Representation

1. **Bag-of-Words (BoW):**

   The Bag of Words (BoW) model is a popular and simple technique used in Natural Language Processing (NLP) and Information Retrieval for text representation. It is used to convert text into a fixed-length vector of word frequencies (or binary values) by treating a document as an unordered collection (bag) of words, disregarding grammar, word order, and sentence structure.

   In BoW, each word is represented as a feature in the vector, and the value in the vector corresponds to either the frequency or presence (binary) of the word in the document. This approach is widely used for tasks such as document classification, sentiment analysis, and topic modeling.

   (a) **Key Concepts of Bag of Words**

   - **Vocabulary Creation:** A vocabulary is created from the entire text corpus. This vocabulary consists of all unique words found in the corpus. The size of the vocabulary is equal to the number of unique words in the corpus.

   - **Vector Representation:** Each document is represented as a vector. The vector's length is equal to the size of the vocabulary. The vector can represent the count (frequency) or presence (binary) of each word in the document.

   - **Unordered Nature:** BoW ignores word order, grammar, and sentence structure. This means that only the presence or frequency of words matters, not the order in which they appear.

   (b) **Steps to Implement the Bag of Words Model**

   i. **Step 1: Tokenization** - Break down the text into individual words (tokens).

ii. **Step 2: Build the Vocabulary** - Identify all unique words in the entire corpus (the collection of documents).

iii. **Step 3: Vectorize the Documents** - Create a vector for each document where each element corresponds to the frequency or binary presence of a word from the vocabulary.

iv. **Step 4: Document Representation** - The document is represented as a vector with the corresponding word frequencies or binary values.

(c) **Example of Bag of Words**

Consider the following 3 documents:

- *"The cat sat on the mat."*
- *"The dog sat on the log."*
- *"The cat and the dog."*
- Step 1: Tokenization

  After tokenizing, the documents become:

  - Document 1: ["The", "cat", "sat", "on", "the", "mat"]
  - Document 2: ["The", "dog", "sat", "on", "the", "log"]
  - Document 3: ["The", "cat", "and", "the", "dog"]
- Step 2: Build the Vocabulary

  The vocabulary is constructed from all unique words in the corpus. After removing stopwords like "the" (optional) and considering only unique words:

$$\text{Vocabulary: } \{cat, sat, on, mat, dog, log, and\}$$

- Step 3: Vectorization

  Each document is now represented as a vector with respect to the vocabulary:

  - Document 1: [1, 1, 1, 1, 0, 0, 0] (The words in Document 1 are "cat", "sat", "on", "mat")
  - Document 2: [0, 1, 1, 0, 1, 1, 0] (The words in Document 2 are "dog", "sat", "on", "log")
  - Document 3: [1, 0, 0, 0, 1, 0, 1] (The words in Document 3 are "cat", "dog", "and")

Appendix A.2.1

2. **Term Frequency-Inverse Document Frequency (TF-IDF):**

TF-IDF is a numerical statistic used to evaluate how important a word is to a document in a collection or corpus. It combines two components:

(a) **Term Frequency (TF)** Term Frequency measures how frequently a term (word) appears in a document. The most common way to compute TF is:

$$TF(t,d) = \frac{\text{Count of term } t \text{ in document } d}{\text{Total number of terms in document } d}$$

Where:

- t = the term (word),
- d = the document.

This gives the relative frequency of the term in the document, with values typically ranging between 0 and 1.

(b) **Inverse Document Frequency (IDF)** Inverse Document Frequency measures how important a word is by considering how rare or common it is across all documents in the corpus. The idea is that words that appear in many documents are less informative.

The formula for IDF is:

$$IDF(t) = \log\left(\frac{D}{df(t)}\right)$$

Where:

- t = the term (word),
- D = the total number of documents in the corpus,
- $df(t)$ = the number of documents containing the term t.

The IDF increases if the term appears in fewer documents and decreases if it appears in more documents. Terms that are too common across all documents (like "the", "is", etc.) will have a low IDF value.

(c) **TF-IDF Calculation** The TF-IDF score of a term is the product of its Term Frequency (TF) and Inverse Document Frequency (IDF):

$$TF-IDF(t,d) = TF(t,d) \times IDF(t)$$

This score reflects the importance of the term t in the document d. A high TF-IDF value means the word is frequent in a particular document but rare in the corpus, making it a potentially important word for that document.

(d) **Example of TF-IDF Calculation** Consider the following small corpus of three documents:

- Document 1: "The cat sat on the mat."
- Document 2: "The dog sat on the log."
- Document 3: "The cat and the dog."
- Step 1: Calculate TF For each word in a document, we calculate the term frequency (TF). Let's compute the TF for the word "cat" in Document 1:

  Total number of words in Document 1 = 6 (["The", "cat", "sat", "on", "the", "mat"])

  Frequency of "cat" in Document 1 = 1

  $$TF(\text{cat, Document 1}) = \frac{1}{6} = 0.167$$

  We do this for each word in each document.

- Step 2: Calculate IDF Next, we calculate the inverse document frequency (IDF) for the term "cat". The term "cat" appears in Document 1 and Document 3, so it appears in 2 documents out of a total of 3.

  $$IDF(\text{cat}) = \log\left(\frac{3}{2}\right) = \log(1.5) \approx 0.176$$

  We repeat this calculation for each word in the corpus.

- Step 3: Calculate TF-IDF Finally, we calculate the TF-IDF for each word in each document by multiplying the TF and IDF values.

  For the word "cat" in Document 1:

  $$TF - IDF(\text{cat, Document 1}) = 0.167 \times 0.176 = 0.029$$

  Repeat this for each word and document in the corpus.

3. **Word Embeddings Word2Vec**:

   Word2Vec is a technique for natural language processing (NLP) that learns vector representations of words in a continuous vector space. These word vectors capture semantic meanings and relationships between words based on their context within a given corpus of text. Unlike traditional methods like one-hot encoding, which represent words as sparse vectors with no inherent meaning, Word2Vec aims to generate dense word embeddings that reflect the meaning of words in a more efficient and meaningful way.

   Word2Vec was developed by Tomas Mikolov and his team at Google in 2013 and has since become one of the most popular models for word embeddings.It is built upon statistical theories, primarily **distributional semantics** and probabilistic models. The key statistical principles in Word2Vec are as follows:

- **Distributional Hypothesis** The foundation of Word2Vec lies in the *distributional hypothesis*, which states:

  "Words that occur in similar contexts tend to have similar meanings."

  This principle implies that a word's meaning can be understood by analyzing its statistical co-occurrence patterns in large text corpora.

- **Language Modeling (Probabilistic Perspective)** Word2Vec uses statistical language modeling to estimate the probability of words based on their context. Given a word w_t, the model predicts:

  – The surrounding context words (*Skip-gram model*).

  – The center word given its surrounding words (*CBOW model*).

- **Skip-gram Model** In the Skip-gram model, the goal is to maximize the probability of context words w_{t+j} given a center word w_t:

$$\text{maximize} \prod_{t=1}^{T} \prod_{-c\leq j\leq c, j\neq 0} P(w_{t+j}|w_t),$$

  where c is the context window size.

- **Continuous Bag of Words (CBOW)** In CBOW, the objective is to predict the center word w_t given its surrounding context:

$$\text{maximize} \prod_{t=1}^{T} P(w_t|\text{context words}).$$

- **Softmax and Negative Sampling** To compute probabilities efficiently, Word2Vec approximates the full softmax computation:

$$P(w_j|w_i) = \frac{\exp(u_j \cdot v_i)}{\sum_{k=1}^{V} \exp(u_k \cdot v_i)},$$

  where u_j and v_i are word vectors, and V is the vocabulary size.

  To reduce computational cost, **negative sampling** is used:

$$\log \sigma(u_j \cdot v_i) + \sum_{k=1}^{N} \mathbb{E}_{w_k \sim P(w)}[\log \sigma(-u_k \cdot v_i)],$$

  where N is the number of negative samples, and σ is the sigmoid function.

- **Matrix Factorization** Word2Vec implicitly factorizes a word-context co-occurrence matrix into low-dimensional word vectors. The embeddings reduce the high-dimensional sparse space into a dense, low-dimensional space that captures semantic relationships.

- **Optimization Using Gradient Descent** The model parameters (word vectors) are optimized using **Stochastic Gradient Descent (SGD)**:

  $$\theta \leftarrow \theta - \eta \frac{\partial L}{\partial \theta},$$

  where η is the learning rate, and L is the loss function.

(a) **Key Concepts of Word2Vec**

- Word Embeddings Word2Vec generates dense, fixed-length vector representations for words. These embeddings are learned such that words with similar meanings or contexts are close together in the vector space.
- Contextual Relationships Word2Vec learns relationships between words based on their context. Words that frequently appear together in similar contexts will have similar vector representations.

(b) **Word2Vec Model Architectures** Word2Vec has two primary architectures for learning word embeddings:

- Continuous Bag of Words (CBOW) In the CBOW model, the goal is to predict a target word (center word) based on a context (surrounding words). For example, given a sentence "The cat sits on the mat", if the target word is "sits", the context words might be ["The", "cat", "on", "the", "mat"]. The CBOW model takes the context words as input and tries to predict the target word. This model is efficient when the corpus is large.
- Skip-Gram Model The Skip-Gram model is the reverse of CBOW. It uses a single word as input (the center word) and tries to predict the context words around it. For the sentence "The cat sits on the mat", if "sits" is the center word, the model will try to predict words like ["The", "cat", "on", "the", "mat"]. The Skip-Gram model works better for smaller corpora and for capturing rare words, as it focuses on predicting multiple context words from a single target word.

(c) **Example of Word2Vec in Action** Consider the sentence "The quick brown fox jumps over the lazy dog." After training a Word2Vec model on a large corpus, the word "quick" might have the following vector representation:

$$\text{quick} = [0.234, 0.512, -0.674, \dots] \quad \text{(a high-dimensional vector)}$$

Now, if you look at words like "fast", "swift", or "speedy", their vectors will be close to "quick" in the vector space, reflecting their semantic similarity.

Similarly, words like "dog" and "cat" will be close in the vector space because they frequently appear together in similar contexts.

Appendix A.2.3

2.2 Sentiment Analysis

2.2.1 Sentiment Analysis Techniques

VADER
The VADER (Valence Aware Dictionary and sEntiment Reasoner) tool is specifically designed for sentiment analysis of social media and news content. It's a lexicon and rule-based tool that's effective in capturing the sentiment intensity of text, especially short texts like news headlines or social media posts. In this project, VADER can be used to analyze news articles and measure their sentiment, which can then be used to predict stock price movements.
VADER provides four scores:

- Positive: The proportion of the text that is positive.
- Neutral: The proportion of the text that is neutral.
- Negative: The proportion of the text that is negative.
- Compound: The aggregated sentiment score, ranging from -1 to 1.

VADER Sentiment Score Algorithm

The following step-by-step algorithm outlines how the VADER (Valence Aware Dictionary and sEntiment Reasoner) calculates sentiment scores:

1. **Input**: A sentence or text string for sentiment analysis.
2. **Tokenization**:
   - Split the input text into individual words (tokens).
   - Preserve punctuation marks and special characters for accurate sentiment detection.
3. **Lexicon Score Look-Up**:
   - For each token, check if it exists in the pre-defined VADER lexicon.
   - **If Found**: Retrieve the associated sentiment score (positive, neutral, or negative).
   - **If Not Found**: Assign a score of `0` (neutral).
4. **Handle Special Cases**:
   - **Negation Words**: Identify tokens like "not," "no," or "never" that modify nearby sentiment-bearing words. Invert the sentiment of the following token(s) within a specific context window (e.g., 3 tokens).
   - **Punctuation**: Adjust the sentiment intensity for tokens followed by exclamation marks (!) or question marks (?).
   - **Capitalization**: If a token is in ALL CAPS, increase its sentiment intensity, assuming it conveys stronger emotion.

5. **Modify Sentiment for Degree Words (Intensity Modifiers)**:
    - Detect words like “very,” “extremely,” or “slightly” (intensifiers or dampeners).
    - Scale the sentiment score of the following token(s) up or down based on the degree word.
6. **Combine Sentiment Scores**:
    - Aggregate the scores of all tokens in the sentence:
        - Positive, Neutral, and Negative sentiment scores are summed separately.
7. **Compute Compound Sentiment Score**:
    - Calculate a **normalized compound score** using the formula:

$$\text{Compound Score} = \frac{\text{Sum of Weighted Sentiment Scores}}{\sqrt{\text{Sum of Squares of Sentiment Scores} + \epsilon}}$$

    - The compound score ranges from -1 (most negative) to $+1$ (most positive).
8. **Output**:
    - **Final Sentiment Scores**:
        - Positive, Neutral, and Negative scores as percentages.
        - Compound Score for overall sentiment evaluation.

Example: *Input Text*: “I absolutely love this product! It’s amazing and works perfectly.”

- **Steps**:
    - Tokenization: [“I,” “absolutely,” “love,” “this,” “product,” “!,” “It’s,” “amazing,” “and,” “works,” “perfectly”].
    - Lexicon Look-Up: Retrieve scores for “love” (+3.0), “amazing” (+4.0), “perfectly” (+2.0).
    - Intensifiers: “absolutely” increases the score of “love” to +4.5.
    - Punctuation: Exclamation mark intensifies the score of the preceding token.
    - Combine Scores: Positive = 10.5, Negative = 0, Neutral = 2 (words without sentiment).
    - Compute Compound Score: Compound = +0.94.
- **Output**: Positive sentiment with a compound score of 0.94.

2.2.2 Implementation

In this implementation, we applied VADER (Valence Aware Dictionary and sEntiment Reasoner) to analyze the sentiment of news articles, which will later serve as input for predicting stock price movements. The steps taken ensure accurate and efficient sentiment analysis of financial news articles, allowing for meaningful data extraction from text.

- 1. Data Preparation
  We start by loading a dataset containing financial news articles. This dataset includes columns such as content (the actual news text) and publishedAt (the publication date of the article). By loading this data into a DataFrame, we can efficiently apply transformations and calculations.
- 2. Sentiment Analyzer Initialization The SentimentIntensityAnalyzer from the nltk.sentiment.vader library is initialized. This tool uses a pre-trained lexicon specifically designed for analyzing the sentiment of short texts, such as news headlines and tweets, making it ideal for financial news sentiment analysis.
- 3. Sentiment Scoring A custom function is defined to extract the compound sentiment score for each news article. The compound score is a single, normalized value ranging from -1 (most negative) to +1 (most positive), summarizing the overall sentiment of the text.This function is then applied to each article in the dataset, generating a sentiment score. We add this score as a new column, sentiment_score, in the DataFrame.
- 4. Sentiment Categorization
  Based on the compound score, the sentiment is categorized as Positive,
  Neutral, or Negative:
  Scores ≥ 0.05 indicate a Positive sentiment.
  Scores ≤ -0.05 indicate a Negative sentiment.
  Scores between -0.05 and 0.05 are labeled Neutral.
  This categorization simplifies interpretation and allows for easier incorporation of sentiment data into models by transforming continuous sentiment scores into categorical labels.

Appendix A.3.1

2.3 Feature Engineering

Feature combining is a powerful technique to enhance your dataset's predictive capacity by creating meaningful relationships and interactions among features. It is essential to experiment with different combinations and validate their effectiveness through model evaluation, as not all combinations will necessarily improve performance. Proper feature engineering, including combining features, is a key step in the success of any predictive modeling project.

- Purpose: This section of code scales the Close prices and the sentiment_score features to a range between 0 and 1.
- Scaling is essential for LSTM models because: LSTMs are sensitive to the scale of the input data. If the input features have different ranges, the model might perform poorly or take longer to converge during training.Scaling improves the optimization process, allowing the model to learn better and faster.

- MinMaxScaler: The MinMaxScaler from the sklearn.preprocessing module is used for this purpose. It transforms each feature by scaling it to a specified range (in this case, 0 to 1). The fit_transform method computes the minimum and maximum values of the features and scales them accordingly.

MinMaxScaler for LSTM Model

In the context of the *NLP-driven Stock Movement Prediction* project, the MinMaxScaler is used to normalize the data before feeding it into an LSTM model. This ensures that the input features are within a specified range, typically between 0 and 1, which improves the performance of the model.

1. **MinMax Scaling Overview**
   MinMax Scaling transforms the data so that each feature is rescaled to a predefined range, typically between 0 and 1. This transformation is essential for neural networks, particularly LSTM models, as they perform better when the input features are on a similar scale.

   $$X_{\text{scaled}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

   Where:

   - X_{scaled} is the scaled value.
   - X is the original value of the feature.
   - X_{min} is the minimum value of the feature.
   - X_{max} is the maximum value of the feature.

2. **Why Use MinMaxScaler in LSTM Models?**

   - **Stability and Convergence:** Neural networks, including LSTM models, often struggle to learn effectively when input features have vastly different scales. By normalizing the data, the model can converge faster and improve accuracy.
   - **Time Series Data:** Stock prices often exhibit wide-ranging fluctuations, making it important to scale the data. This ensures that the model treats all features equally, preventing bias towards larger values such as stock prices.

3. **Applying MinMaxScaler in Stock Prediction** When using LSTM for stock prediction, you typically deal with time-series data like stock prices and sentiment scores. Here is how MinMaxScaler is applied to normalize these features:

4. **Normalizing Stock Prices and Sentiment Scores**

   - **Stock Prices:** MinMaxScaler can be used to scale the stock prices, ensuring that all values are transformed into a range between 0 and 1.
   - **Sentiment Scores:** Sentiment scores, which can range from negative to positive values, are also normalized to ensure that they are on the same scale as the stock prices.

   Scaling Data for LSTM

- Input Data: After scaling, you can split your data into input (X) and output (y) for training the LSTM model.
- Inverse Scaling: Once the model predicts stock movement, you can reverse the scaling to interpret the results on the original scale.

Appendix**??**

2.4 Statistical Models

Statistical models in NLP refer to methods that use statistical and probabilistic principles to analyze and model natural language data. These models aim to represent the underlying patterns and structures of language, and are fundamental for tasks such as language modeling, machine translation, text classification, and part-of-speech tagging. They rely heavily on large corpora to estimate probabilities and distributions of linguistic units (such as words, phrases, and sentences) and their interactions.

The primary idea behind statistical models in NLP is to use data-driven approaches that learn language representations and structure from real-world text data. Below is an overview of the key statistical models used in NLP:

2.4.1 N-gram Models

N-gram models are one of the simplest and most widely used statistical models in NLP. They predict the next word in a sequence based on the previous $n-1$ words, where n is the number of words considered in the context. These models are particularly useful for tasks such as language modeling and text generation.

- **Unigram Model (1-gram)** This model predicts a word based only on its individual probability, without considering its context. It assumes that all words are independent of each other.

  $$P(w_i) = P(w_i \mid w_{i-1}, w_{i-2}, \dots, w_1)$$

  where $P(w_i)$ is simply the probability of the word w_i.

- **Bigram Model (2-gram)** This model predicts the next word based on the previous word.

  $$P(w_i \mid w_{i-1}) = \frac{\text{Count}(w_{i-1}, w_i)}{\text{Count}(w_{i-1})}$$

- **Trigram Model (3-gram)** This model predicts the next word based on the previous two words, and so on for higher-order n-grams.

  N-gram models are easy to implement, but they have limitations. As n increases, the model becomes more computationally expensive, and data sparsity becomes a problem.

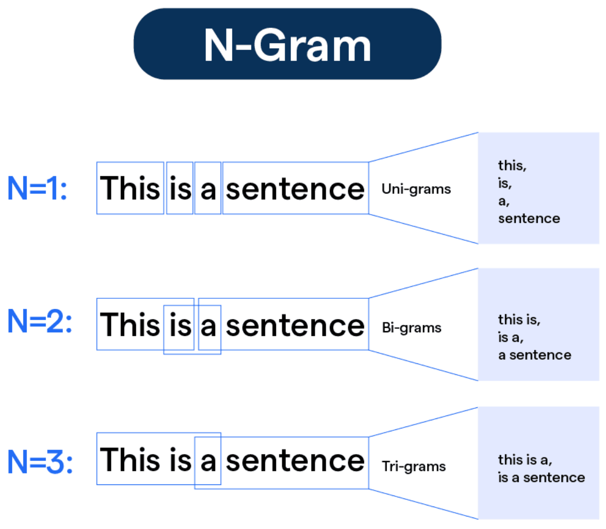

Figure 2.2: N-gram model.

- **Applications**
  - Language Modeling: Predicts the next word in a sequence based on previous words.
  - Speech Recognition: Helps transcribe spoken language into text by predicting word sequences.
  - Text Generation: Used to generate coherent text by predicting the next word.
  - Spelling Correction: Detects spelling errors by checking the context of words.
  - Information Retrieval: Assists in ranking documents based on word sequences.

2.4.2 Hidden Markov Models (HMM)

Hidden Markov Models (HMMs) are probabilistic models that assume a system can be in one of a set of states, and that the states are hidden (i.e., not directly observable). HMMs are often used in tasks like part-of-speech tagging and speech recognition.

An HMM consists of:

- A set of hidden states $S = \{s_1, s_2, \dots, s_n\}$.
- An observable output (e.g., words or speech signals) $O = \{o_1, o_2, \dots, o_m\}$.
- Transition probabilities $P(s_t \mid s_{t-1})$ that define the probability of transitioning between states.
- Emission probabilities $P(o_t \mid s_t)$ that define the probability of observing an output given a state.

Key Problem In an HMM, given a sequence of observations, the goal is to infer the sequence of hidden states. This is typically solved using the Viterbi algorithm, which finds the most likely sequence of hidden states given the observed data.

Applications

- Part-of-Speech Tagging: Labels words with their respective parts of speech (e.g., noun, verb).
- Named Entity Recognition (NER): Identifies entities like names, dates, and locations in text.
- Speech Recognition: Used to map spoken words to text by modeling phoneme sequences.
- Handwriting Recognition: Converts handwritten text into digital text.
- Bioinformatics

2.4.3 Naive Bayes Classifier

The Naive Bayes classifier is a simple probabilistic classifier based on Bayes' theorem with the naive assumption that the features (words) are conditionally independent given the class label. Despite this simplifying assumption, Naive Bayes can perform surprisingly well in many NLP tasks, particularly in text classification tasks like spam detection or sentiment analysis.

The formula for Naive Bayes is:

$$P(C \mid X) = \frac{P(X)}{P(C)} \prod_{i=1}^{n} P(x_i \mid C)$$

where:

- C is the class (e.g., spam or not spam),
- X is the feature vector (e.g., a set of words),
- $P(C)$ is the prior probability of the class,
- $P(x_i \mid C)$ is the likelihood of observing the word x_i given the class C,
- $P(X)$ is the evidence or normalizing constant.

Applications

- Spam Detection: Classifies emails as spam or not based on word frequencies.
- Sentiment Analysis: Classifies text (e.g., reviews) into positive, negative, or neutral sentiment.
- Document Classification: Categorizes documents (e.g., news articles) into predefined topics.
- Medical Diagnosis: Classifies patient data based on symptoms or test results.

- Language Identification: Detects the language of a text based on word patterns.

2.4.4 Statistical Machine Translation (SMT)

Statistical Machine Translation (SMT) is a technique used for translating text from one language to another using statistical models, which are trained on parallel corpora. Unlike rule-based or neural machine translation systems, SMT relies on statistical models to perform the translation task by analyzing large amounts of bilingual text data.

1. **Overview of Statistical Machine Translation (SMT)**

   SMT involves two primary components:

   - **Translation Model**: This model is responsible for mapping words or phrases from the source language to the target language. It is typically trained on large bilingual corpora, where corresponding sentences in both languages are provided.
   - **Language Model**: The language model assesses the fluency of the translated output by predicting the likelihood of word sequences in the target language. It helps ensure that the translation respects the grammar and natural flow of the target language.

2. **Steps in Statistical Machine Translation**

   (a) **Data Preparation**: A large dataset of sentence pairs in two languages is required (e.g., English-Spanish, English-French). Each sentence in one language has a corresponding translation in the other language.

   (b) **Word Alignment**: Word alignment is the process of identifying which words in the source sentence correspond to words in the target sentence. This is often done using algorithms like IBM Models or HMM (Hidden Markov Models).

   (c) **Model Training**:

   - **Translation Model**: This model is trained to align source and target language words and phrases. It is typically based on phrases or words, with probabilities associated with how well a word or phrase in the source language can be translated to the target language.
   - **Language Model**: A statistical model is trained on the target language alone to estimate the probability of word sequences in the target language (e.g., n-grams).

   (d) **Decoding**: The translation is generated by finding the most likely sequence of words in the target language, based on the word alignments and the language model probabilities.

   (e) **Post-processing**: Additional steps like reordering words or smoothing probabilities are performed to improve the quality of the translation.

3. **Types of Statistical Machine Translation**

- **Word-based SMT**: This approach treats the translation process as a word-for-word mapping from the source to the target language. It uses word alignments to translate each word individually, but it may struggle with idiomatic expressions or complex sentence structures.
- **Phrase-based SMT**: This approach considers sequences of words (phrases) rather than individual words. It improves upon word-based SMT by capturing phrase-level translations, which are more natural and fluent than word-for-word translations. The phrase-based model uses a larger translation table and statistical methods to align entire phrases.
- **Syntax-based SMT**: Syntax-based SMT uses syntactic structures (like parse trees) to align and translate sentences. It aims to preserve the grammatical structure of the sentence in both the source and target languages.
- **Hierarchical SMT**: Hierarchical SMT further enhances phrase-based translation by considering hierarchical patterns of phrases and sub-phrases, leading to better fluency and more accurate translations.

4. **Example of SMT Process**

   For instance, let's say we are translating an English sentence to French:

   English sentence: "I am learning machine translation."

   - **Data Preparation**: We have a parallel corpus with sentences like:

     English: "I am learning machine translation."
     French: "J'apprends la traduction automatique."

   - **Word Alignment**: The system identifies corresponding words in both languages:
     - "I" → "Je"
     - "am" → "suis"
     - "learning" → "apprends"
     - "machine translation" → "traduction automatique"
   - **Model Training**: The system learns probabilities for each translation pair from the corpus, such as how likely "machine translation" is to be translated as "traduction automatique".
   - **Decoding**: The system uses the trained translation and language models to generate a translated sentence:

     Output: "Je suis en train d'apprendre la traduction automatique."

Chapter 3

Application

3.1 Application of Word2Vec for stock movement prediction using LSTM model

1. **Input Data** The LSTM model requires the following input data:

   - **Historical Stock Prices:** Previous day's closing prices, technical indicators (e.g., moving averages, RSI, Bollinger Bands), and other relevant historical data.

   - **News Sentiment Scores:** Aggregated daily sentiment scores derived from news articles related to the stock.

   The target variable can be:

   - **Price Movement (Up/Down):** For classification tasks.

2. **Preprocessing** Before training the LSTM model, data preprocessing is essential:

   - **Normalization:** Features are scaled to a specific range (e.g., 0 to 1) using techniques like Min-Max scaling or standardization to ensure stable training.

   - **Sequence Generation:** The input data is reshaped into sequences of fixed length (e.g., 10 days). Each sequence represents a time window of historical data used to predict the future value.

3. **Training** The LSTM model is trained using the following steps:

   - **Loss Function:**

     – Classification: Binary Cross-Entropy loss is used to assess the model's ability to classify price movements.

   - **Validation Split:**

     – A portion of the training data is set aside for validation to monitor the model's performance and prevent overfitting.

4. **Prediction** Once the model is trained, it can be used to predict future stock movements. Given a sequence of recent historical data, the model generates predictions for the next time step.

Appendix A.3.2

3.2 Application of Naive Bayes for Classifying News Sentiment of Reliance Stock

1. **Overview of the Problem** The goal of this project is to predict the sentiment (positive or negative) of news articles related to Reliance stock. These sentiment scores can then be used to predict stock movements, offering valuable insights for financial decision-making.

   The dataset contains the following columns:

   - Date: The publication date of the news.
   - News: The headline or content of the news related to Reliance stock.
   - Sentiment: The sentiment label, either "Positive" or "Negative," indicating the nature of the news.

   Using the Naive Bayes Classifier, we analyze the news articles to classify their sentiment based on the text content.

2. **Steps in Applying Naive Bayes to the Dataset**

   - Data Preprocessing: Remove missing values and clean the text (e.g., lowercase, remove stop words). Prepare a dataset with columns: Date, News, and Sentiment.
   - Feature Extraction: Convert textual data into numerical format using TF-IDF Vectorizer to represent the importance of words in the dataset.
   - Data Splitting: Divide the dataset into training (70%) and testing (30%) sets to build and evaluate the model.
   - Model Training: Train the Multinomial Naive Bayes classifier using the TF-IDF-transformed training data to learn sentiment classification.
   - Prediction: Use the trained model to predict sentiment labels (Positive/Negative) for the test set.
   - Evaluation: Evaluate the model's performance using metrics like Precision, Recall, and F1-Score to assess accuracy and reliability.

   Appendix A.3.3

3.3 Application of N-grams in Auto complete

1. Text Preprocessing: The corpus is tokenized into sentences and words, all converted to lowercase for consistency.
2. Bigram Generation: The script generates bigrams (nltk.util.ngrams) from the tokenized text, representing sequences of two consecutive words.
3. Bigram Model: A defaultdict stores each word as a key and maps it to a Counter object containing its probable next words with their frequencies.
4. Prediction: The model predicts the most likely next words for a given input word using the bigram frequencies.
5. Autocomplete: The autocomplete() function takes a partial sentence as input and uses the bigram model to suggest the next words.

Appendix A.3.4

3.4 Machine Translation Services Using Statistical Machine Translation (SMT)

Statistical Machine Translation (SMT) is a technique in computational linguistics that translates text from one language to another using statistical models based on the analysis of large amounts of bilingual text data. SMT works by leveraging aligned bilingual corpora (parallel text data) to learn translation rules and generate translations based on probabilistic models. While SMT has largely been overtaken by more advanced approaches like Neural Machine Translation (NMT), it remains an important historical method for machine translation and continues to be used in certain cases.

1. **Key Concepts of SMT in Machine Translation**
    - **Parallel Corpus**: SMT relies on large collections of parallel corpora—text data where the same content exists in two or more languages. The parallel corpus serves as the foundation for training the statistical models.
    - **Word Alignment**: SMT algorithms first identify how words in one language correspond to words in another language. This is called word alignment. Common techniques used for alignment include IBM models (e.g., IBM Model 1) and Hidden Markov Models (HMM). These alignments are then used to extract translation probabilities between words and phrases.
    - **Translation Model**: The translation model stores probabilities of how words or phrases in the source language translate into words or phrases in the target language. The higher the probability, the more likely a translation will be produced.
    - **Language Model**: The language model in SMT helps ensure that the translated output is grammatically correct and fluent. This model is usually trained on large amounts

of text in the target language and estimates the likelihood of different word sequences occurring in that language.

- **Decoding**: The decoding process is where the actual translation happens. The system uses both the translation model and the language model to generate the most likely translation. A decoding algorithm, such as the Viterbi algorithm or beam search, is used to find the best translation.

2. **Key Steps in SMT**

   - Data Preparation
     - **Collecting Parallel Corpora**: The first step in building an SMT system is to gather parallel corpora (sentences in both the source and target languages).
     - **Preprocessing**: Text in the source and target languages may need to be cleaned and tokenized before training the model.
   - Word Alignment Using algorithms like IBM Model 1 or HMM, the system identifies word correspondences between the two languages. This allows the system to create translation probabilities for each word or phrase.
   - Training the Translation Model The translation model is built by calculating probabilities of translating one word or phrase into another. For example, if the phrase “machine translation” occurs frequently in both English and French, the model will learn that the translation “traduction automatique” has a high probability of being correct.
   - Building the Language Model A language model is trained on a large corpus of text in the target language to ensure the fluency and grammatical correctness of the output. Typically, n-grams are used for building this model.
   - Decoding and Translation When a new sentence is given for translation, the system uses the translation model and language model to produce the most probable translation. This is done using algorithms like beam search or the Viterbi algorithm.

Appendix A.3.5

Chapter 4

Results and Conclusion

4.1 Results

1. Word2Vec

   - Confusion Matrix The confusion matrix for the classification model is as follows:

     $$\begin{bmatrix} 22 & 89 \\ 33 & 127 \end{bmatrix}$$

     Where:

     - The first row corresponds to the true class 0, with 22 correct predictions (True Negatives) and 89 incorrect predictions (False Positives).
     - The second row corresponds to the true class 1, with 33 incorrect predictions (False Negatives) and 127 correct predictions (True Positives).

   - Classification Report The classification report provides detailed performance metrics for each class:

Class	Precision	Recall	F1-score	Support
0	0.40	0.20	0.27	111
1	0.59	0.79	0.68	160
Accuracy			0.55	271
Macro avg	0.49	0.50	0.47	271
Weighted avg	0.51	0.55	0.51	271

   - Performance Metrics The overall model evaluation metrics are as follows:

     - **Accuracy**: 0.5498
     - **Precision**: 0.588
     - **Recall**: 0.794

- **F1 Score**: 0.676

These metrics help in understanding the model's effectiveness in making predictions on the test dataset.

2. **Naive Bayes**

(a) Confusion Matrix The confusion matrix for the classification model is as follows:

$$\begin{bmatrix} 112 & 136 \\ 55 & 205 \end{bmatrix}$$

Where:

- The first row corresponds to the true class 0, with 112 correct predictions (True Negatives) and 136 incorrect predictions (False Positives).
- The second row corresponds to the true class 1, with 55 incorrect predictions (False Negatives) and 205 correct predictions (True Positives).

(b) Classification Report The classification report provides detailed performance metrics for each class:

Class	Precision	Recall	F1-score	Support
0.0	0.67	0.45	0.54	248
1.0	0.60	0.79	0.68	260
Accuracy			0.62	508
Macro avg	0.64	0.62	0.61	508
Weighted avg	0.64	0.62	0.61	508

(c) Performance Metrics The overall model evaluation metrics are as follows:

- **Accuracy**: 0.624
- **Precision**: 0.601
- **Recall**: 0.788
- **F1 Score**: 0.682

These metrics provide insights into the model's performance on the test dataset and its ability to handle different classes effectively.

Insights and Future Improvements

Insights:

- **Effectiveness of Statistical Models:** Statistical models, such as Naive Bayes and N-gram models, provide valuable insights into the core of text processing tasks like sentiment analysis, language modeling, and text generation. These models, although relatively simple, often perform surprisingly well in many real-world applications.

  - *Naive Bayes* is particularly effective for text classification tasks where feature independence can be assumed, such as spam detection or sentiment analysis.
  - *N-gram models* are effective for generating text and making predictions based on the preceding words, making them useful for applications like autocomplete or machine translation.

- **Trade-offs in Performance:** While simpler models like Naive Bayes perform quickly and are easier to implement, they may not capture the complexities of language as effectively as more advanced models, such as those based on deep learning. For example, Naive Bayes relies on the assumption of feature independence, which may not hold true in all cases, leading to suboptimal results in certain contexts.
- **Handling Sparse Data:** In N-gram models, data sparsity becomes an issue as the value of "n" increases. The larger the n-gram size, the more likely it is that certain sequences of words do not appear in the training data, making it harder for the model to make accurate predictions. This is a limitation when dealing with larger corpora or when attempting to model rare word sequences.
- **Model Interpretability:** Statistical models like Naive Bayes are relatively easy to interpret compared to deep learning models. For tasks where interpretability is key (e.g., financial applications or medical diagnostics), statistical models can be advantageous.

4.2 Conclusions

The application of statistical models in Natural Language Processing (NLP) plays a significant role in addressing a wide array of challenges related to language understanding and processing. Throughout this project, we explored the various statistical techniques used in NLP, including N-gram models, Hidden Markov Models (HMM), and Naive Bayes classifiers. Each of these methods offers unique advantages and limitations when applied to real-world NLP tasks.

4.2.1 Summary of Key Findings

- **N-gram Models:** N-gram models, particularly Unigram, Bigram, and Trigram models, are foundational in tasks like language modeling, text generation, and text classification. Despite their simplicity, they suffer from issues like data sparsity and the inability to capture long-range dependencies in text. However, they remain highly effective in many practical applications, especially when used in combination with smoothing techniques and other enhancements.
- **Hidden Markov Models (HMM):** HMMs are powerful for sequential data tasks such as part-of-speech tagging, named entity recognition, and speech recognition. They excel at modeling probabilistic state transitions, although they are limited by the assumption of independence between the current state and previous states.
- **Naive Bayes Classifier:** Naive Bayes, despite its strong independence assumption, performs well in classification tasks such as spam detection and sentiment analysis. It is

computationally efficient and easy to implement, making it a popular choice for text classification problems. However, its reliance on feature independence often leads to suboptimal performance when dependencies between features are significant.

4.2.2 Challenges and Limitations

- **Data Sparsity:** In statistical models like N-grams, especially with higher-order models (e.g., trigrams or 4-grams), the issue of data sparsity arises. As the "n" value increases, the likelihood of encountering previously unseen word combinations increases, leading to poor model performance. This problem is particularly prominent in language tasks with less frequent words or specialized vocabulary.
- **Independence Assumption in Naive Bayes:** Naive Bayes models rely on the assumption that features (i.e., words) are conditionally independent given the class. This assumption is often unrealistic in natural language, where the occurrence of one word can heavily influence the likelihood of others. This can lead to suboptimal performance, especially in tasks involving complex sentence structures or context-dependent meanings.
- **Handling Ambiguity:** Natural language is inherently ambiguous, with multiple meanings for the same word depending on the context. Statistical models often struggle to disambiguate words or phrases in these situations. For example, the word "bank" could refer to a financial institution or the side of a river, and without contextual information, models might misinterpret such cases.
- **Quality of Training Data:** The performance of statistical models heavily depends on the quality and quantity of the training data. If the training corpus is noisy or unrepresentative of the real-world data, the model's ability to generalize to unseen data may be compromised. Moreover, biased or incomplete data can lead to biased or inaccurate predictions.
- **Out-of-Vocabulary (OOV) Words:** Statistical models, especially N-gram models, face challenges with out-of-vocabulary words that do not appear in the training data. While smoothing techniques can mitigate this issue, they cannot fully address the problem of handling rare or unseen words, which can significantly affect model performance in real-world applications.
- **Scalability:** As the size of the corpus grows, statistical models such as N-grams require more memory and computational power to store and process the increasing number of word combinations or features. This makes scaling these models to large corpora or real-time applications challenging.
- **Inability to Capture Long-Range Dependencies:** Statistical models like Naive Bayes and N-grams typically fail to capture long-range dependencies in text. For instance, the relationship between words or phrases spread across sentences or paragraphs may not be effectively modeled. This is especially problematic for tasks requiring a deep understanding of context, such as machine translation or question answering.
- **Limited to Syntactic Analysis:** While statistical models excel at syntactic-level tasks like

classification, they struggle with tasks requiring semantic understanding. They focus on patterns of words but may fail to comprehend the deeper meaning behind the text, which is essential for tasks like summarization, sentiment analysis, and contextual text generation.

- **Difficulty with Idiomatic Expressions:** Idioms, slang, and domain-specific terms can present challenges for statistical models. These expressions often do not follow standard syntactic or semantic rules, and statistical models that rely on patterns may fail to interpret them correctly. For instance, "kick the bucket" is understood as a colloquial expression for death, but a statistical model might mistakenly interpret it as literal.
- **Overfitting:** Statistical models like Naive Bayes are prone to overfitting, especially when the training data is small or unrepresentative. This happens when the model becomes too tailored to the training data and fails to generalize to new, unseen data, resulting in poor performance on test datasets.

4.2.3 Future Improvements

- **Incorporating Deep Learning Models:** While statistical models provide a strong foundation, incorporating deep learning models (like Recurrent Neural Networks, Transformers, and BERT) can improve performance in more complex tasks such as context-aware sentiment analysis, machine translation, and text generation. These models can capture long-range dependencies and complex word relationships that statistical models may struggle with.
- **Data Augmentation:** In statistical models like Naive Bayes and N-grams, having a large and diverse dataset is key to improving model performance. Data augmentation techniques, such as paraphrasing or generating synthetic data, could be implemented to increase the size and variety of the training dataset, improving model robustness.
- **Feature Engineering:** Advanced feature extraction techniques like word embeddings (Word2Vec, GloVe) or sentence embeddings (BERT) can be used to represent text data more effectively, capturing semantic relationships between words and improving model accuracy. These features could be used in conjunction with statistical models to enhance performance.
- **Domain-Specific Adaptation:** Statistical models can be improved by adapting them to specific domains. For instance, for stock news sentiment analysis, using domain-specific corpora to train the model can help the model learn industry-specific language and terminologies, resulting in better predictions.
- **Integration with Larger Systems:** While individual models are useful, integrating statistical models into larger systems, such as recommendation engines, chatbots, or predictive analytics systems, can unlock more value. Combining NLP models with other machine learning techniques (e.g., reinforcement learning) could lead to more intelligent, context-aware systems.

Appendix A

Appendix

A.1 Data Preprocessing

A.1.1 Lower Casing

```python
#1. Lower_Casing
import pandas as pd
data=pd.read_csv(r"C:\Users\marat\OneDrive\Desktop\2nd
   year\project\Reliance_news_data2.csv")
df=data
df['content'] = df['content'].astype(str)
df['content']=df['content'].apply(lambda x: x.lower())
print(df.head())
```

Listing A.1: Python code for lower casing

A.1.2 Tokenization

```python
#2. Remove_Punctuations

import string
string.punctuation

exclude=string.punctuation

def rem_pun(text):
    for char in exclude:
        text=text.replace(char,'')
    return text

import re
def remove_tags(raw_text):
    cle_text=re.sub(re.compile('<.*?>'),'',raw_text)
    return cle_text

df['content']=df['content'].apply(rem_pun)
print(df.head())
```

Listing A.2: Python code for tokenization

A.1.3 Stemming

```
#4. Stemming
from nltk.stem.porter import PorterStemmer


def stem_tokens(tokens):
  stemmer = PorterStemmer()
  stemmed_tokens = [stemmer.stem(token) for token in tokens]
  return stemmed_tokens

def stem_dataframe(df):
  df['stemmed_contents'] =
     df['tokenized_contents'].apply(stem_tokens)
  return df

# Stem the tokens
df = stem_dataframe(df)
print(df.head())
```

Listing A.3: Python code for stemming

A.1.4 Lemmatization

```


import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet


def get_wordnet_pos(word):
  tag = nltk.pos_tag([word])[0][1][0].upper()
  tag_dict = {"J": wordnet.ADJ,
              "N": wordnet.NOUN,
              "V": wordnet.VERB,
              "R": wordnet.ADV}
  return tag_dict.get(tag, wordnet.NOUN)

def lemmatize_tokens(tokens):
  lemmatizer = WordNetLemmatizer()
  lemmatized_tokens = [lemmatizer.lemmatize(token,
     get_wordnet_pos(token)) for token in tokens]
  return lemmatized_tokens

def lemmatize_dataframe(df):
  df['lemmatized_contents'] =
     df['tokenized_contents'].apply(lemmatize_tokens)
  return df

# Lemmatize the tokens
df = lemmatize_dataframe(df)
print(df.head())
```

Listing A.4: Python code for lemmatization

A.2 Text Representation

A.2.1 Bag of Words

```

import sklearn
from sklearn.feature_extraction.text import CountVectorizer



cv = CountVectorizer()
bow = cv.fit_transform(combined_list)  # Replace with df.iloc[1, 8]
    as per your original code

# Create a DataFrame from the BoW matrix
bow_df = pd.DataFrame(bow.toarray(),
    columns=cv.get_feature_names_out())

# Vocabulary
vocab = cv.vocabulary_

# Print Vocabulary
print("Vocabulary:")
print(vocab)

# Display the Bag of Words DataFrame
print("\nBag of Words DataFrame:")
print(bow_df)

vocab_summary = pd.DataFrame([vocab]).rename(index={0: 'Vocabulary'})

```

```python
final_output = pd.concat([vocab_summary, bow_df], ignore_index=False)

print("\nFinal Output (BoW DataFrame with Vocabulary):")
print(final_output)
```

Listing A.5: Python code for Bag of Words

A.2.2 TF-IDF

```python
#3. Term Frequency and Inverse Document Frequency (TF-IDF)
from sklearn.feature_extraction.text import TfidfVectorizer

tf = TfidfVectorizer()
txt_fit = tf.fit(df.iloc[1,6])
txt_transform = txt_fit.transform(df.iloc[1,6])
idf = tf.idf_
print(dict(zip(txt_fit.get_feature_names_out(), idf)))

#
tokenized_sentences = df['tokenized_contents'].tolist()
txt_fit = tf.fit(combined_list)
txt_transform = txt_fit.transform(combined_list)
idf = tf.idf_
print(dict(zip(txt_fit.get_feature_names_out(), idf)))
```

Listing A.6: Python code for TF-IDF

A.2.3 Word2Vec

```python
import pandas as pd
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize
import nltk


model = Word2Vec(sentences=tokenized_sentences, vector_size=100,
    window=5, min_count=1, workers=4)

# Example: Get the vector for a specific word
word_vector = model.wv['reliance']  # Replace 'document' with the
    word you want to check
print("Vector for 'reliance':")
print(word_vector)

# Example: Finding similar words
similar_words = model.wv.most_similar('reliance', topn=5)
print("\nMost similar words to 'reliance':")
print(similar_words)
```

Listing A.7: Python code for Word2Vec

A.2.4 Scaling of Features

```python

# Scale features
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)

# Reshape data for LSTM
sequence_length = 10  # Use 10 days of data to predict movement
X_sequences = []
y_sequences = []

for i in range(sequence_length, len(X_scaled)):
    X_sequences.append(X_scaled[i-sequence_length:i])
    y_sequences.append(y.values[i])

X_sequences = np.array(X_sequences)
y_sequences = np.array(y_sequences)
```

Listing A.8: Python code for scaling of features

A.3 Modelling

A.3.1 Calculation of Sentiment Score

```python
from nltk.sentiment.vader import SentimentIntensityAnalyzer
import pandas as pd
import nltk

# Download VADER lexicon if not already downloaded
#nltk.download('vader_lexicon')

# Initialize the SentimentIntensityAnalyzer
sia = SentimentIntensityAnalyzer()

# Calculate sentiment scores
news_data["Sentiment_Score"] = news_data["News"].apply(lambda x:
    sia.polarity_scores(x)["compound"])

# Display the dataframe with sentiment scores
news_data.head()
```

Listing A.9: Python code for calculation of sentiment score

A.3.2 LSTM Model

```python
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
from gensim.models import Word2Vec
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Load data
#  Replace with your data paths
news_data = pd.read_csv(r"C:\Users\marat\OneDrive\Desktop\2nd year\NLP-driven-stock-movement-prediction\Reliance_News.csv")  # Assumes news_data has 'sentiment_score' and 'date'
stock_data = pd.read_csv(r"C:\Users\marat\OneDrive\Desktop\2nd year\NLP-driven-stock-movement-prediction\reliance_stock_data.csv")  # Assumes stock_data has 'date', 'close', and 'movement'

# Preprocess stock data
# Create the Movement column
stock_data["Movement"] = (stock_data["Close"].diff() > 0).astype(int)


stock_data.head()


# Merge news and stock data
merged_data = pd.merge(news_data, stock_data, on='Date')
merged_data = merged_data.sort_values('Date')

# Word2Vec embedding for news headlines
def preprocess_text(text):
    return text.lower().split()

merged_data['processed_headline'] = merged_data['News'].apply(preprocess_text)

# Train Word2Vec model
w2v_model = Word2Vec(sentences=merged_data['processed_headline'], vector_size=100, window=5, min_count=1, workers=4)

# Create averaged Word2Vec vectors for headlines
def vectorize_headline(headline):
    vectors = [w2v_model.wv[word] for word in headline if word in
       w2v_model.wv]
    return np.mean(vectors, axis=0) if vectors else np.zeros(100)

merged_data['headline_vector'] =
   merged_data['processed_headline'].apply(vectorize_headline)

# Feature selection
features = ['headline_vector']
feature_vectors = np.array(merged_data['headline_vector'].tolist())
scaled_close = MinMaxScaler().fit_transform(merged_data[['Close']])
X = np.hstack([feature_vectors])
y = merged_data['Movement']

# Reshape data for LSTM
sequence_length = 10  # Use 10 days of data to predict movement
X_sequences = []
y_sequences = []

for i in range(sequence_length, len(X)):
    X_sequences.append(X[i-sequence_length:i])
    y_sequences.append(y.values[i])

X_sequences = np.array(X_sequences)
y_sequences = np.array(y_sequences)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_sequences,
   y_sequences, test_size=0.2, random_state=42)

# Build the LSTM model
model = Sequential([
    LSTM(64, return_sequences=True, input_shape=(X_train.shape[1],
       X_train.shape[2])),
    Dropout(0.2),
    LSTM(64),
    Dropout(0.2),
    Dense(1, activation='sigmoid')  # Sigmoid for binary
       classification
])

model.compile(optimizer='adam', loss='binary_crossentropy',
   metrics=['accuracy'])

class_weight = {0: 1.5, 1: 1}  # Adjust the weight for class 0 to be
   higher
```

```
history =model.fit(X_train, y_train, epochs=20, batch_size=32,
   validation_data=(X_test, y_test), class_weight=class_weight)


# Evaluate the model
eval_loss, eval_accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {eval_loss}, Test Accuracy: {eval_accuracy}")

# Save the model
model.save('stock_movement_lstm_word2vec_model.h5')

# Predict
y_pred = (model.predict(X_test) > 0.5).astype(int)

# Evaluation metrics
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("Classification Report:")
print(classification_report(y_test, y_pred))
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
print(f"Precision: {precision_score(y_test, y_pred)}")
print(f"Recall: {recall_score(y_test, y_pred)}")
print(f"F1 Score: {f1_score(y_test, y_pred)}")
```

Listing A.10: Python code for LSTM model

A.3.3 Naive Bayes Model

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report

df=pd.read_csv(r"C:\Users\marat\OneDrive\Desktop\2nd
   year\NLP-driven-stock-movement-prediction\Reliance_News.csv")
# Check for NaN values
print(df.isna().sum())

# Remove rows with any NaN values
df = df.dropna()
# 2. Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df['News'],
   df['sentiment'], test_size=0.3, random_state=42)

# 3. Convert text data into numerical format using TF-IDF Vectorizer
vectorizer = TfidfVectorizer(stop_words='english')
```

```
X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# 4. Train a Naive Bayes model
nb_model = MultinomialNB()
nb_model.fit(X_train_tfidf, y_train)

# 5. Make predictions on the test set
y_pred = nb_model.predict(X_test_tfidf)

# 6. Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred,
   target_names=["Negative", "Positive"]))
```

Listing A.11: Python code for Naive Bayes model

A.3.4 N-gram model

```
import nltk
from nltk.util import ngrams
from collections import defaultdict, Counter

# Sample text corpus
corpus = """
Artificial intelligence is transforming the world.
The future of AI is bright and full of opportunities.
Machine learning and deep learning are subsets of AI.
AI is revolutionizing industries like healthcare and finance.
"""

# Step 1: Tokenize the text into sentences and words
sentences = nltk.sent_tokenize(corpus)
tokenized_sentences = [nltk.word_tokenize(sentence.lower()) for
   sentence in sentences]

# Step 2: Generate bigrams
bigrams = []
for sentence in tokenized_sentences:
    bigrams.extend(list(ngrams(sentence, 2)))

# Step 3: Create a dictionary to map words to their probable next
   words
bigram_model = defaultdict(Counter)
for word1, word2 in bigrams:
    bigram_model[word1][word2] += 1

# Step 4: Predict next word based on the current word
```

```python
def predict_next_word(current_word, n_predictions=3):
    if current_word in bigram_model:
        probable_words = bigram_model[current_word].most_common(n_predictions)
        return [word for word, count in probable_words]
    else:
        return []

# Step 5: Autocomplete function
def autocomplete(prompt, n_predictions=3):
    last_word = prompt.strip().split()[-1].lower()
    suggestions = predict_next_word(last_word, n_predictions)
    return suggestions

# Test the autocomplete
prompt = "AI is"
suggestions = autocomplete(prompt, n_predictions=3)

print(f"Prompt: '{prompt}'")
print("Suggestions:", suggestions)
```

Listing A.12: Python code for N-gram model

A.3.5 SMT model

```python
import numpy as np
from collections import defaultdict

# Toy Parallel Corpus (English-French sentences)
english_sentences = ["I am learning machine translation",
                     "Statistical machine translation is important",
                     "Machine learning models are powerful"]
french_sentences = ["Je suis en train d'apprendre la traduction automatique",
                    "La traduction automatique statistique est importante",
                    "Les mod les d'apprentissage automatique sont puissants"]

# Step 1: Create word alignment (for simplicity, we'll align each word directly based on occurrence)
def create_word_alignment(english_sentences, french_sentences):
    word_alignments = defaultdict(lambda: defaultdict(int))

    for eng, fr in zip(english_sentences, french_sentences):
        eng_words = eng.split()
        fr_words = fr.split()

        for eng_word in eng_words:
            for fr_word in fr_words:
                word_alignments[eng_word][fr_word] += 1

    # Normalize the word alignments to probabilities
    translation_probs = defaultdict(lambda: defaultdict(float))
    for eng_word, fr_map in word_alignments.items():
        total_count = sum(fr_map.values())
        for fr_word, count in fr_map.items():
            translation_probs[eng_word][fr_word] = count / total_count

    return translation_probs

# Step 2: Train the translation model (word alignment model)
translation_probs = create_word_alignment(english_sentences, french_sentences)

# Step 3: Translate a new sentence (test)
def translate_sentence(sentence, translation_probs):
    eng_words = sentence.split()
    translation = []

    for word in eng_words:
        if word in translation_probs:
            # Get the French word with the highest probability
            best_translation = max(translation_probs[word], key=translation_probs[word].get)
            translation.append(best_translation)
        else:
            translation.append(word)  # If the word is not in the model, keep it unchanged

    return ’ ’.join(translation)

# Step 4: Test translation
test_sentence = "I am learning machine translation"
translated_sentence = translate_sentence(test_sentence, translation_probs)

print(f"Original sentence: {test_sentence}")
print(f"Translated sentence: {translated_sentence}")
```

Listing A.13: Python code for SMT model

—

—

—

A.4 Hardware and Software Specifications

The project was implemented using the following specifications:

- **Hardware:**
  - Processor: Ryzen 5, 3.4 GHz
  - RAM: 16 GB
  - GPU: NVIDIA GEFORCE GTX
- **Software:**
  - Programming Language: Python 3.10
  - IDE: Jupyter Notebook
  - Libraries: TensorFlow, NLTK, SpaCy, scikit-learn, pandas

The jupyter notebook with all codes and the dataset used for building model are available at " Github/Harshal Marathe".

Bibliography

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. An empirical investigation of statistical significance in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 995–1005, 2012.

Paul S Jacobs. Joining statistics with nlp for text categorization. In *third conference on Applied natural language processing*, pages 178–185, 1992.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Pearson, 3rd edition, 2023.

Joakim Nivre. On statistical methods in natural language processing. In *Proceedings of the 13th Nordic Conference of Computational Linguistics (NODALIDA 2001)*, 2001.